

# Designing quantum chemistry codes for next-generation supercomputers

Jeff Hammond (jhammond@anl.gov)

Argonne National Laboratory – Leadership Computing Facility

CECAM – 5 September 2011



# Abstract (for posterity)

I will discuss the design of quantum chemistry codes for next-generation supercomputers in light of experience with a wide variety of architectures, including Blue Gene/P, Cray XE, Infiniband clusters, and NVIDIA GPUs. Programming models and algorithms that maximize performance will be described in detail. On the primary topics of this workshop — fault tolerance and energy efficiency — I will focus on the most obvious sources of faults (programmer error) and energy inefficiency (suboptimal algorithms). I will also describe my approach to optimizing quantum chemistry codes for Blue Gene/Q, which is by far the most energy efficient supercomputer to date ([green500.org](http://green500.org)).

# Does power matter?

August 29, 2011

## Exascale: Power Is Not the Problem!

Andrew Jones, Vice-President of HPC Services and Consulting, Numerical Algorithms Group

---

According to Jones, the real issue is delivering science to justify whatever the power budget is and that human expertise in algorithms and programming is more expensive than power.

## If power doesn't matter



# Cray X1E

- True globally accessible memory (pointer = node + address).
- Vector machines do very well at dense linear algebra.
- Cray compilers genuinely good at optimization.
- No caches, just load from main memory in  $O(1)$  cycles.
- Who needs MPI?

According to Buddy Bland, scientists were upset when it was decommissioned in favor of Jaguar.

18 TF for 1+ MW was justified when Cray XT3/4 could do 10x better.

# Power matters I

Hardware not designed for HPC, more suitable for Word and FPS.

Two trends in HPC processor architecture:

- 1** More-of-the-same (MOTS): more cores, more caches, more bells+whistles.
- 2** Back-to-the-future (BTTF): vectors - SIMD ala FPU and SIMT ala GPU.

In reality, we have both MOTS and BTTF in every machine; integration varies. Tight integration is FPU in the cores ala BG; loose integration is Cray XK6 (CPU+GPU), although CPU part is same idea as other short-vector cores but with more NUMA.

Welcome to the future: the kitchen sink approach to architecture design.

# Power matters II

Power efficiency forces us to use “parallelism all the way down”...

Green500 summary:

- 1** Blue Gene/Q (2097.19 MF/W)
- 4** Intel+NVIDIA (958.35 MF/W)
- 14** POWER7 (565.97 MF/W)
- 19** Cell-based (458.33 MF/W)
- 21** Intel-only (441.18 MF/W)

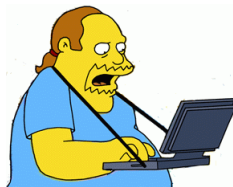
You can't do anything on BGQ without significant parallelism:

10 PF = 48 r = 49,152 n = 786,432 c = 3,145,728 t

3M threads is not business as usual but it's not like CPU+GPU is a programmer's dream.

# Science matters, right?

We are not getting vector machines back because of science...



Scientific simulation has to find ways to leverage the commercial computing fad of the day (GUI, FPS, smartphone,...) unless someone comes up with 10-100x the money for hardware.



# Power-efficient tensor algorithms

The real work is done by:

Edgar Solomonik

Devin Matthews

Martin Schatz

# Power-efficient algorithms

The most efficient way to minimize power consumption is to do the same thing in less time.

Given the idle draw of most hardware, you cannot come close to 50% reduction from a power-aware algorithm, but many scientific algorithms can run 2x faster with superior math.

Power-aware runtimes, etc. are pointless unless they are running algorithms that are more than big-O notation optimal: both  $10^4 n$  and  $n$  are  $O(n)$ . Power-efficient  $O(n^3)$  is asinine compared to power-hungry  $O(n)$ .

*How much time/money/power is wasted every year because Top500 doesn't allow Strassen?*

# Tensor Contraction Engine

What does it do?

- 1** GUI input quantum many-body theory e.g. CCSD.
- 2** Operator specification of theory.
- 3** Apply Wick's theory to transform operator expressions into array expressions.
- 4** Transform input array expression to operation tree using many types of optimization.
- 5** *Produce Fortran+Global Arrays+NXTVAL implementation.*

Developer can intercept at various stages to modify theory, algorithm or implementation.

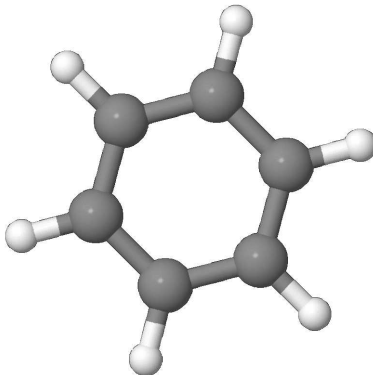
# The practical TCE – Success stories

- First parallel implementation of many (most) CC methods.
- First truly generic CC code (not string-based):  
 $\{\text{RHF, ROHF, UHF}\} \times \text{CC}\{\text{SD, SDT, SDTQ}\} \times \{T/\Lambda, \text{EOM, LR/QR}\}$
- Most of the largest calculations of their kind employ TCE:  
CR-EOMCCSD(T), CCSD-LR  $\alpha$ , CCSD-QR  $\beta$ , CCSDT-LR  $\alpha$
- Reduces implementation time for new methods from years to hours, TCE codes are easy to verify.

*Significant hand-tuning by Karol Kowalski and others at PNNL was required to make TCE run efficiently and scale to 1000 processors and beyond.*

# Before TCE

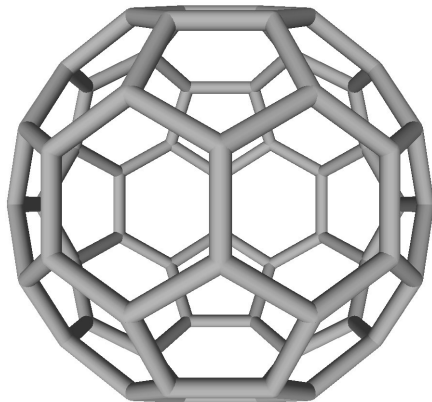
CCSD/aug-cc-pVDZ – 192 b.f. – days on 1 processor



Benzene is close to crossover point between *small* and *large*.

# Linear response polarizability

CCSD/Z3POL – 1080 b.f. – 40 hours on 1024 processors

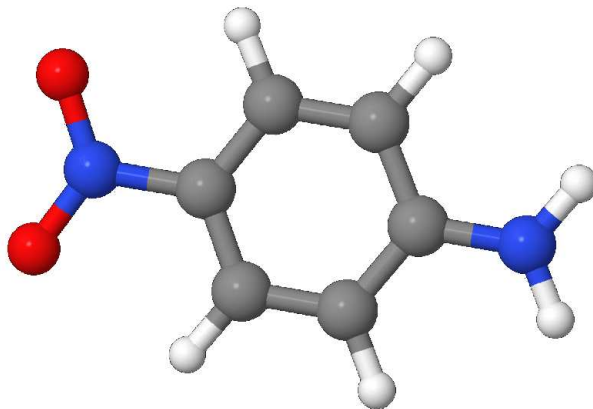


This problem is 20,000 times larger on the computer than benzene.

*J. Chem. Phys.* **129**, 226101 (2008).

# Quadratic response hyperpolarizability

CCSD/d-aug-cc-pVTZ – 812 b.f. – 20 hours on 1024 processors



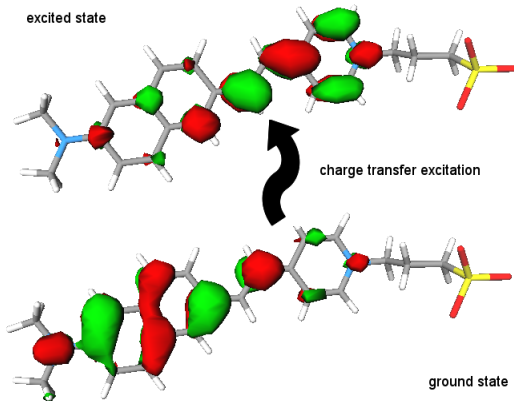
Lower levels of theory are not reliable for this system.

*J. Chem. Phys.* **130**, 194108 (2009).



# Charge-transfer excited-states of biomolecules

CR-EOMCCSD(T)/6-31G\* – 584 b.f. – 1 hour on 256 cores

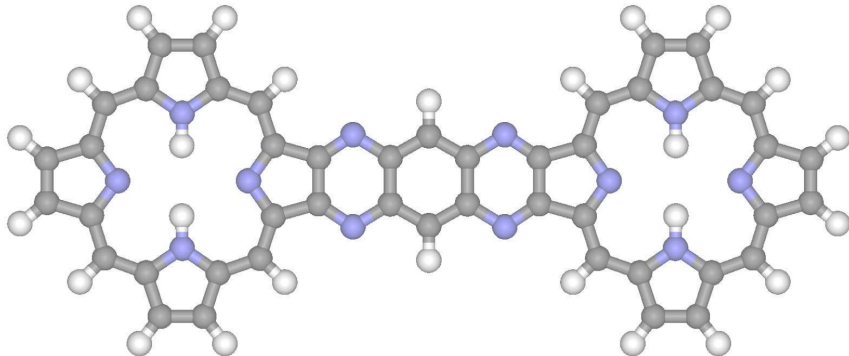


Lower levels of theory are wildly incorrect for this system.



# Excited-state calculation of conjugated arrays

CR-EOMCCSD(T)/6-31+G\* – 1096 b.f. – 15 hours on 1024 cores



*J. Chem. Phys.* **132**, 154103 (2010).

Even bigger systems done in Kowalski, et al., *JCTC* **7**, 2200 (2011).



# So what's the problem?

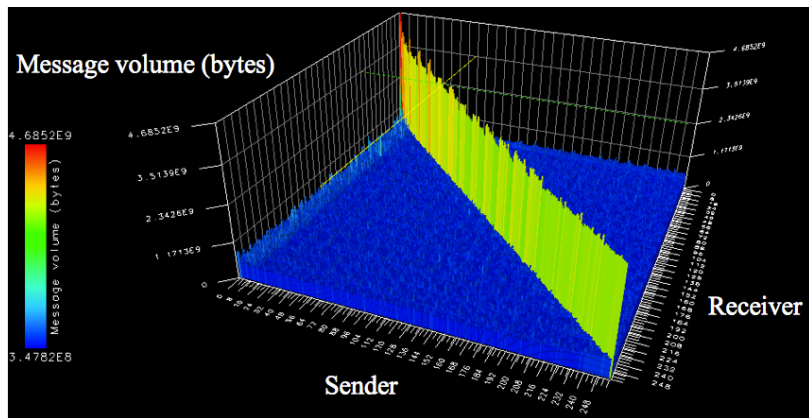
Pseudocode for  $R_{i,j}^{a,b} = R_{i,j}^{c,d} * V_{a,b}^{c,d}$ :

```
for i,j in occupied blocks:
  for a,b in virtual blocks:
    for c,d in virtual blocks:
      if symmetry_criteria(i,j,a,b,c,d):
        if dynamic_load_balancer(me):
          Get block t(i,j,c,d) from T
          Permute t(i,j,c,d)
          Get block v(a,b,c,d) from V
          Permute v(a,b,c,d)
          r(i,j,c,d) += t(i,j,c,d) * v(a,b,c,d)
    Permute r(i,j,a,b)
  Accumulate r(i,j,a,b) block to R
```

# So what's the problem?

- DLB doesn't consider distance of data or reuse.
- No reuse of permuted local patches:  
e.g. `Permute t(i,j,c,d)` called  $O(N_{tiles}^2)$  times.
- Permute is an evil version of STREAM:  
 $\forall i,j,k,l : A_{ijkl} = B_{P(ijkl)}$ .
- Get is effectively two-sided.
- DLB requires remote atomics (in general, not in HW).
- All-connect topology is bad on a torus...

# TCE algorithm topology



# So what's the solution?

Tensor contractions are like big parallel matrix multiplication operations, so why not reuse those algorithms (Cannon, SUMMA)?

- + Use collective communication instead of one-sided, which is optimal on Blue Gene (2x BW in bcast).
- + Minimal local operations (computation and data reshuffling).
- Symmetry is hard to handle (DLB and one-sided were used by TCE for a reason).
- If we do matrix multiplication globally, we have to do permutation globally, generating alltoall patterns.

The critical challenge is to solve the symmetry issue. . .

Note: Fast collectives are not just a BG feature. One can use systolic collectives on a Cray if the scheduler provides contiguous partitions.

# Summary

Until someone comes up with a numerical robust reduced-scaling CC algorithm, the best thing to do is make the existing algorithms run faster.

Preliminary results suggest we can reduce time-to-solution by 2-5x, reduce memory usage by 2x and make efficient use of the most power-efficient architecture available (Blue Gene).

We expect this to increase power-efficiency by 2-5x on the same hardware and as much as 20x relative to COTS clusters that are *designed to run NWChem*.

# OSPRI

New **O**ne-**S**ided **P**RImitives for Global Arrays and Other  
One-sided Models

Most of the real work was done by Sreeram Potluri.

# The PGAS programming model

Use of Global Arrays (GA) by NWChem driven by programmer productivity:

- hides complexity of distributed data
- easy dynamic load-balancing
- ameliorates local memory limitations
- wrappers to math libraries (ScaLAPACK, PeiGS)

ARMCI emerged later as the communication runtime component within Global Arrays.

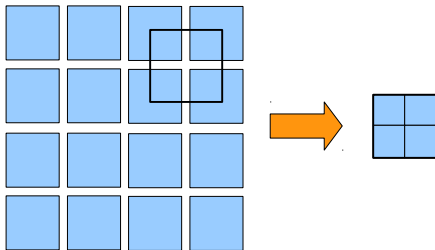
The NWChem project started before MPI was available and certainly before a mature set of libraries were built upon it.



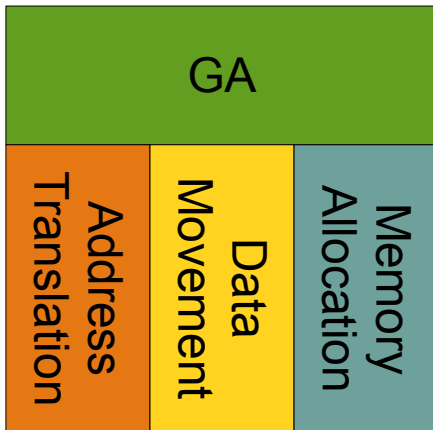
# Global Arrays behavior

GA\_Get arguments: handle, *global* indices, pointer to target buffer

- 1** translate global indices to rank plus local indices
- 2** issue remote get operations to each rank
- 3** remote packing of non-contiguous data
- 4** packed buffer arrives at caller
- 5** local buffer assembled



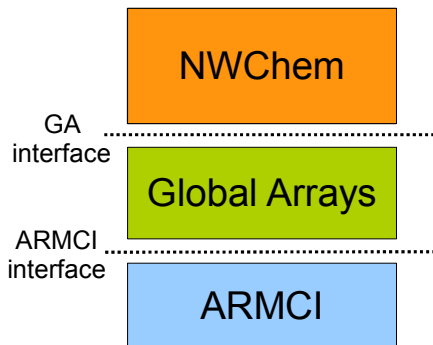
# Global Arrays components



Parallel math capability is orthogonal because ScaLAPACK and PeIGS use MPI.

This abstraction is not implemented effectively.

# Overview



- Put, Get, Acc (CSI)
- Rmw (scalar ints)
- Fence, Sync

# Modern System Design Points



- Torus topology (2.5-6D)
- Lightweight Linux(-like) OS
- Low-latency, high-injection NIC
- RDMA with offloading
- MPI+OpenMP, other hybrids
- HW GAS support
- Reliable, connectionless networks
- HW collectives

Some systems (BG) provide HW active-messages.

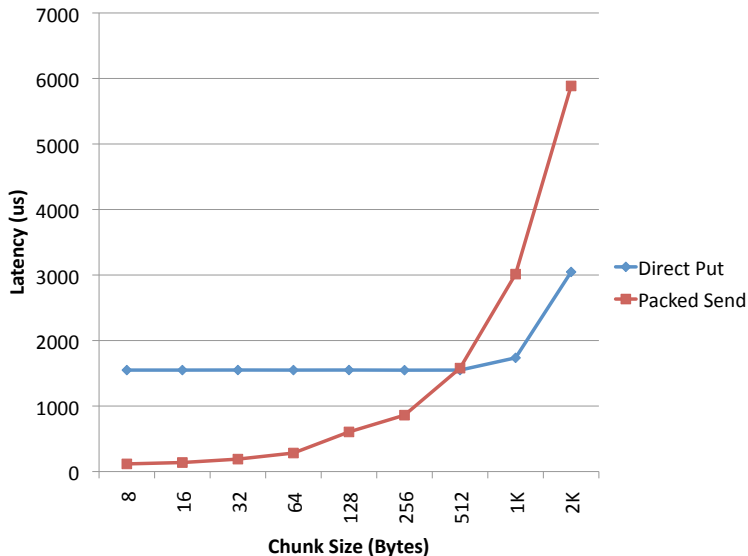
# Design by benchmarking

Want quantitate answers for:

- Accumulate: interrupts versus polling (CHT)?
- Non-contiguous: direct versus packing?
- Local: DMA versus memcpy?
- Local: DMA contention?
- Remote completion: ack versus flush?
- Packing: inline versus handoff (CHT only)?
- Buffering: heap or internal?
- Buffering: handling resource exhaustion.

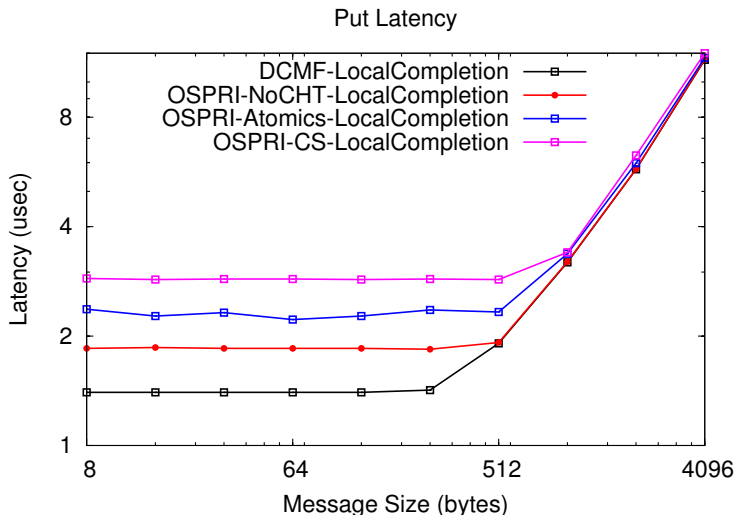
Whenever the answer is usage-dependant, we use a tunable runtime parameter.

# Crossover for packing



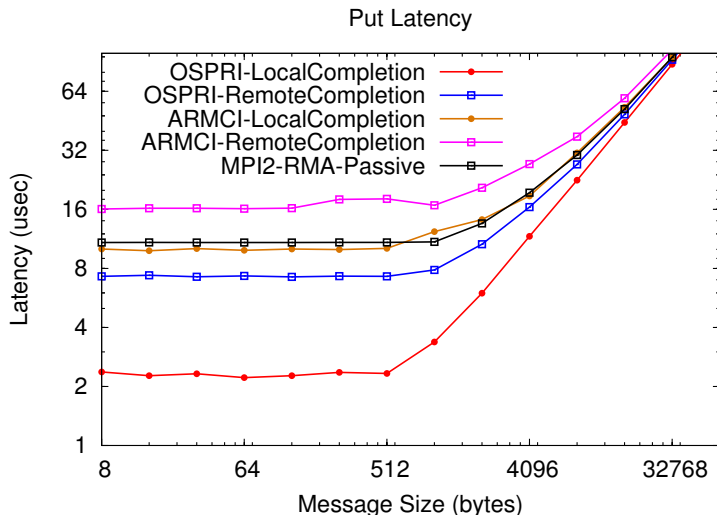
# Overhead for thread-safety

DCMF critical sections are a heavy hammer.



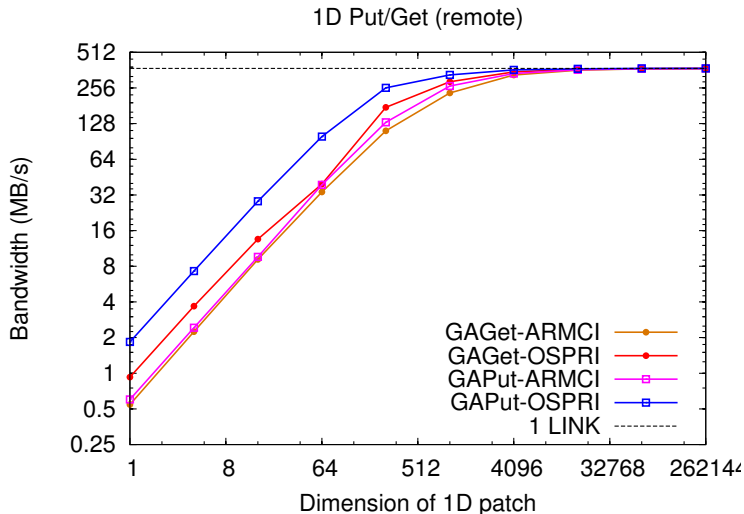
# Versus other runtimes

These ARMCI results much improved over original implementation.

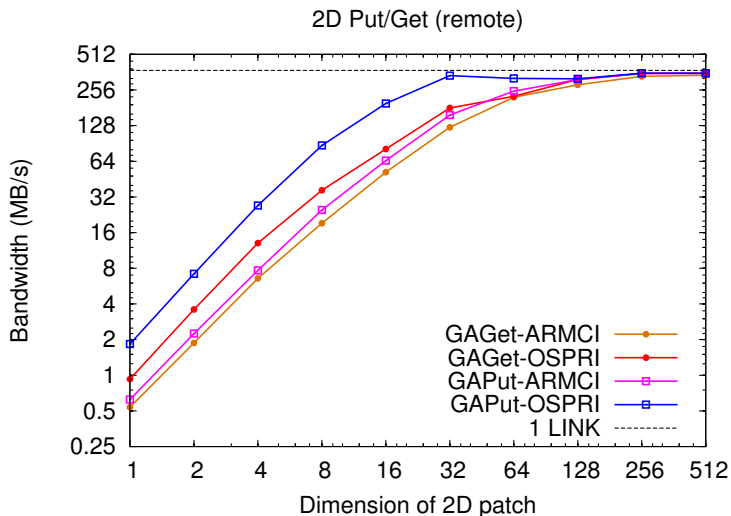




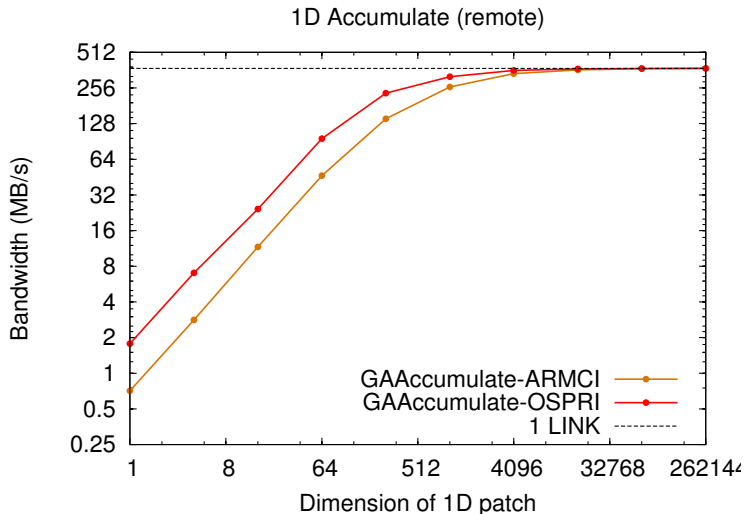
# GA Put/Get — 1D remote



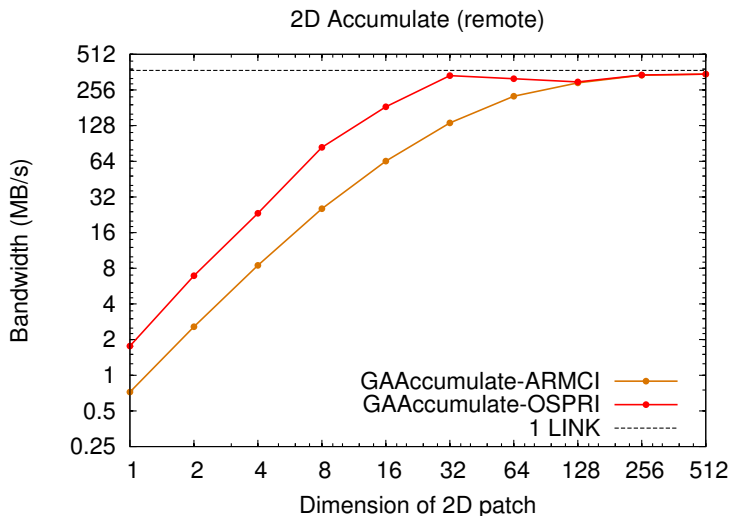
# GA Put/Get — 2D remote



# GA Acc — 1D remote



# GA Acc — 2D remote

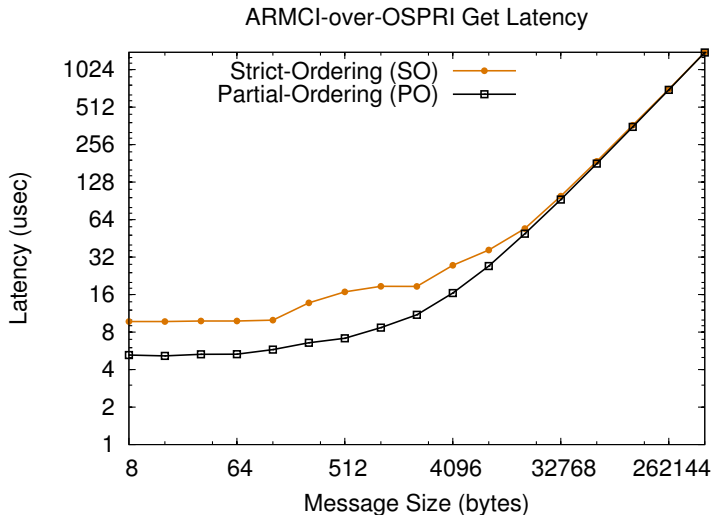


# Ordering semantics

- ARMCI provides location consistency (load-store ordering).
- NWChem and other GA apps don't need this.
- Only necessary for conflicting access.
- Metadata to detect conflicts from sender is prohibitive.
- Push consistency up stack to user: flush as needed.
- Cost of ordering grows with system.
- Ordering prohibitive when network doesn't provide it.

OSPRI design philosophy is to be soft on consistency, let the user (or higher level library) worry about it.

# Effect of ordering semantics



# Programming Blue Gene/Q

# Things I can tell you

I cannot clarify or elaborate on any of the following statements.  
Please do not ask.

- 16-core nodes, 4-way SMT (up to 64 threads per node)
- 16 Gb memory per node
- 1.6 GHz
- 204.8 GFLOP/s per node
- 48 racks = 49,152 nodes = 786,432 cores
- Argonne system is 10 PF.



# Things IBM has said publicly



*Welcome to the* **2011** **19**

19th Annual  
**Symposium on High-Performance Interconnects**  
The international forum where the high-performance computing and high-speed networking communities meet.

[Home](#) | [Program](#) | [Keynotes](#) | [Tutorials](#) | [Registration](#) | [Attendees](#) | [Committees](#) | [Sponsors](#) | [Contact](#)

## Keynote 1

### Speaker

Philip Heidelberger, Research  
Staff Member, IBM

### Title

The IBM Blue Gene/Q  
Interconnection Network and  
Message Unit




The following quotes are taken from the abstract alone.

Source: <http://www.hoti.org/hoti19/keynotes/>.

- “The chip has 11 ports; each port can transmit data at 2 GB/s and simultaneously receive at 2 GB/s for a total bandwidth of 44 GB/s.”
- “The network consists of a five dimensional compute node torus with a subset of compute nodes using the 11th port to connect to I/O nodes.”
- “Blue Gene/Q machine has approximately 46 times the bisection bandwidth than that of a first generation Blue Gene/L machine with the same number of nodes.”
- “A single network supports point-to-point and collective traffic such as MPI all reduces at near link bandwidth.”
- “The collectives can be over an entire partition or any rectangular subset of a partition. The network provides bit reproducible, single pass floating point collectives.”

# Things IBM has said publicly



Power.ORG   
Webinar Series

## HPC - Innovative technologies for power management based on Power Architecture

**Dr. Luigi Brochard, Distinguished Engineer, IBM Deep  
Computing Worldwide Solution Architect**

**March 29, 2011**

**Celebrating 20<sup>th</sup>  
Anniversary of  
Power Architecture**

# Processor

Source:

[http://www.power.org/events/PowerWebinar-03-29-11/...IBM\\_March\\_29\\_Webinar\\_-\\_Dr.\\_Luigi.pdf](http://www.power.org/events/PowerWebinar-03-29-11/...IBM_March_29_Webinar_-_Dr._Luigi.pdf).

- Collective and barrier networks are embedded in 5-D torus.
- Integer and floating-point addition support in collective network.
- SIMD floating point unit (8 flop/clock).
- Speculative multithreading and transactional memory support with 32 MB of speculative state.
- 17th Processor core for system functions.

# Programming Blue Gene/Q – Software

All of our efforts are portable to other systems except vector intrinsics, which are never portable (e.g. SSE).

- MPI ranks: 48K (1/n), 800K (1/c) or 3M (1/t)?  
Program for 1/n, accept 1/c for unthreadable legacy codes.
- Combine task and data parallelism in threading model.
  - Pthread+OpenMP interoperability is a challenge on BGP and in general.
  - TBB porting activities are in-progress. We have TBB on BGP and POWER7.
- Vectorization done via libraries and intrinsics.

Emphasize separation of internode and intranode parallelism, leading to reuse of internode components on heterogeneous systems.

# Programming Blue Gene/Q – Algorithms

- Hierarchical instead of flat: FMM, tree-codes.
- Asynchronous everything (leverage async. coll.).
- Topology-aware distributed data-structures (e.g. GPAW on BGP).
- Move away from DLB and 1-sided but use OSPRI when necessary.

# Acknowledgments

**Argonne:** Pavan Balaji, Jim Dinan, Eugene DePrince

**Ohio State:** Sreeram Potluri

**IBM:** Brian Smith and Mike Blocksome

**Jülich:** Ivo Kabadshow and Holger Dachsel

**PNNL:** Sriram Krishnamoorthy

**UOregon:** Sameer Shende and Allen Malony

**Texas:** Devan Matthews, Jack Poulson, Martin Schatz, Robert van de Geijn

**Berkeley:** Edgar Solomonik, (Jim Demmel)

# What is TCE?

- 1** NWChem users know it as the coupled-cluster module that supports a kitchen sink of methods.
- 2** NWChem developers know it as the Python program that generates item 1.
- 3** People in this room probably know it as a multi-institutional collaboration that resulted in item 2 (among other things).



# Summary of TCE module

```
http://cloc.sourceforge.net v 1.53  T=30.0 s
```

```
-----  
Language      files  blank  comment      code  
-----  
Fortran 77    11451   1004   115129  2824724  
-----  
SUM:          11451   1004   115129  2824724  
-----
```

Only <25 KLOC are hand-written; ~100 KLOC is utility code following TCE data-parallel template.

# My thesis work

```
http://cloc.sourceforge.net v 1.53  T=13.0 s
```

```
-----  
Language      files  blank  comment   code  
-----
```

```
Fortran 77      5757      0    29098  983284  
-----
```

```
SUM:            5757      0    29098  983284  
-----
```

Total does not include  $\sim 1\text{M}$  LOC that was reused (EOM).

**CCSD quadratic response hyperpolarizability was derived, implemented and verified during a two week trip to PNNL. Over 100 KLOC were “written” in under an hour.**

# The practical TCE – Limitations

What does it NOT do?

- Relies upon external (read: hand-written) implementations of many procedures.
- Hand-written procedures define underlying data representation.
- Does not effectively reuse code (TCE needs own runtime).  
Of course, 4M LOC in F77 could be 4K LOC in C++.
- Ignores some obvious abstraction layers and hierarchical parallelism.

None of these shortcomings are intrinsic!

An instantiation of TCE is limited to the set of code transformations known to the implementer.

# The practical TCE – Performance analysis

Performance TCE in NWChem cannot be understood independent of GA programming model.

- GA couldn't do block sparse so TCE does its own indexing. Table lookups are/were a bottleneck.
- Suboptimal data representation leads to nonlocal communication.
- Single-level tiling isn't ideal.
- Lack of abstraction for kernel prevents optimization; e.g. tensor permutations significant portion of wall time.

Time does not permit me to quantify performance issues.

Can *all* hand-optimizations can be back-ported into TCE?

Is this necessary? What are the challenges of an embedded DSL?

# HPC circa 2012

Systems coming online in 2012 will have 200K+ cores with significant node-level parallelism both in the processor(s) and the NIC (e.g. Cray Gemini has 48 ports).

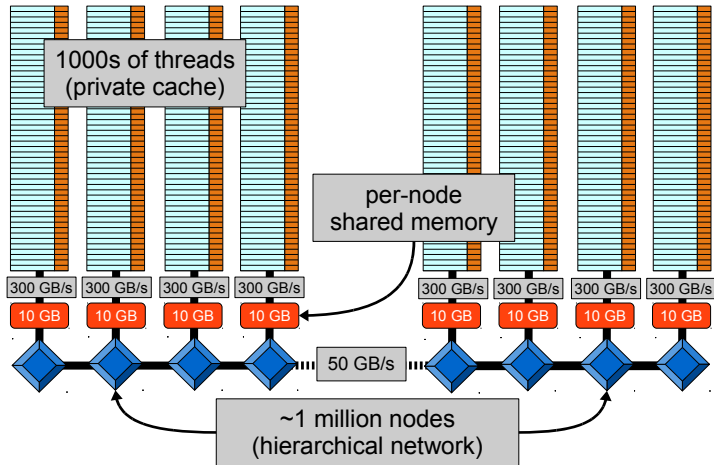
- BlueWaters (PERCS): 37,500+ sockets, 8 cores per socket.
- Mira (Blue Gene/Q): 49,152 nodes, 1 16-core CPU per node.
- Titan (Cray XK): 12,160 nodes, 1 AMD Bulldozer CPU and 1 NVIDIA Kepler GPU per node.

[Details from NCSA, Wikipedia and Buddy Bland's public slides.]

Node counts not increasing relative to current systems, so node-level parallelism is our primary challenge.

Process-only parallelism is not optimal for any of these machines. TCE 2.0 must address heterogeneity.

# Exascale Architecture



# Coupled-cluster theory

$$|CC\rangle = \exp(T)|0\rangle$$

$$T = T_1 + T_2 + \cdots + T_n \quad (n \ll N)$$

$$T_1 = \sum_{ia} t_i^a \hat{a}_a^\dagger \hat{a}_i$$

$$T_2 = \sum_{ijab} t_{ij}^{ab} \hat{a}_a^\dagger \hat{a}_b^\dagger \hat{a}_j \hat{a}_i$$

$$\begin{aligned} |\Psi_{CCD}\rangle &= \exp(T_2)|\Psi_{HF}\rangle \\ &= (1 + T_2 + T_2^2)|\Psi_{HF}\rangle \end{aligned}$$

$$\begin{aligned} |\Psi_{CCSD}\rangle &= \exp(T_1 + T_2)|\Psi_{HF}\rangle \\ &= (1 + T_1 + \cdots + T_1^4 + T_2 + T_2^2 + T_1 T_2 + T_1^2 T_2)|\Psi_{HF}\rangle \end{aligned}$$

# Coupled cluster (CCD) implementation

$$R_{ij}^{ab} = V_{ij}^{ab} + P(ia, jb) \left[ T_{ij}^{ae} I_e^b - T_{im}^{ab} I_j^m + \frac{1}{2} V_{ef}^{ab} T_{ij}^{ef} + \right. \\ \left. \frac{1}{2} T_{mn}^{ab} I_{ij}^{mn} - T_{mj}^{ae} I_{ie}^{mb} - I_{ie}^{ma} T_{mj}^{eb} + (2T_{mi}^{ea} - T_{im}^{ea}) I_{ej}^{mb} \right]$$

$$I_b^a = (-2V_{eb}^{mn} + V_{be}^{mn}) T_{mn}^{ea}$$

$$I_j^i = (2V_{ef}^{mi} - V_{ef}^{im}) T_{mj}^{ef}$$

$$I_{kl}^{ij} = V_{kl}^{ij} + V_{ef}^{ij} T_{kl}^{ef}$$

$$I_{jb}^{ia} = V_{jb}^{ia} - \frac{1}{2} V_{eb}^{im} T_{jm}^{ea}$$

$$I_{bj}^{ia} = V_{bj}^{ia} + V_{be}^{im} (T_{mj}^{ea} - \frac{1}{2} T_{mj}^{ae}) - \frac{1}{2} V_{be}^{mi} T_{mj}^{ae}$$

Tensor contractions currently implemented as GEMM plus PERMUTE.



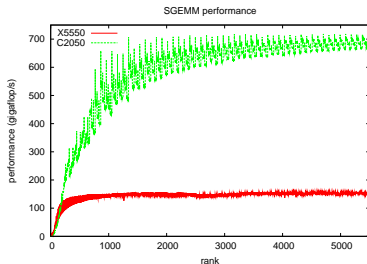
# Hardware Details

	CPU		GPU	
	X5550	2 X5550	C1060	C2050
processor speed (MHz)	2660	2660	1300	1150
memory bandwidth (GB/s)	32	64	102	144
memory speed (MHz)	1066	1066	800	1500
ECC available	yes	yes	no	yes
SP peak (GF)	85.1	170.2	933	1030
DP peak (GF)	42.6	83.2	78	515
power usage (W)	95	190	188	238

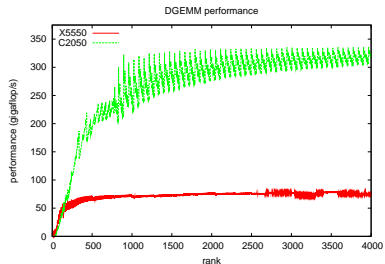
Note that power consumption is apples-to-oranges since CPU does not include DRAM, whereas GPU does.

# Relative Performance of GEMM

GPU versus SMP CPU (8 threads):



CPU = 156.2 GF  
GPU = 717.6 GF



CPU = 79.2 GF  
GPU = 335.6 GF

We expect roughly 4-5 times speedup based upon this evaluation because GEMM *should* be 90% of the execution time.

# CPU/GPU CCD

	Iteration time in seconds					
	our DP code			X5550		
	C2050	C1060	X5550	Molpro	TCE	GAMESS
C <sub>8</sub> H <sub>10</sub>	0.3	0.8	1.3	2.3	5.1	6.2
C <sub>10</sub> H <sub>8</sub>	0.5	1.5	2.5	4.8	10.6	12.7
C <sub>10</sub> H <sub>12</sub>	0.8	2.5	3.5	7.1	16.2	19.7
C <sub>12</sub> H <sub>14</sub>	2.0	7.1	10.0	17.6	42.0	57.7
C <sub>14</sub> H <sub>10</sub>	2.7	10.2	13.9	29.9	59.5	78.5
C <sub>14</sub> H <sub>16</sub>	4.5	16.7	21.6	41.5	90.2	129.3
C <sub>20</sub>	8.8	29.9	40.3	103.0	166.3	238.9
C <sub>16</sub> H <sub>18</sub>	10.5	35.9	50.2	83.3	190.8	279.5
C <sub>18</sub> H <sub>12</sub>	12.7	42.2	50.3	111.8	218.4	329.4
C <sub>18</sub> H <sub>20</sub>	20.1	73.0	86.6	157.4	372.1	555.5

Our algorithm is most similar to GAMESS and does  $\sim 4$  times the flops as Molpro.

# CPU+GPU CCSD

	Iteration time (s)						
	Hybrid	CPU	Molpro	NWChem	PSI3	TCE	GAMESS
C <sub>8</sub> H <sub>10</sub>	0.6	1.4	2.4	3.6	7.9	8.4	7.2
C <sub>10</sub> H <sub>8</sub>	0.9	2.6	5.1	8.2	17.9	16.8	15.3
C <sub>10</sub> H <sub>12</sub>	1.4	4.1	7.2	11.3	23.6	25.2	23.6
C <sub>12</sub> H <sub>14</sub>	3.3	11.1	19.0	29.4	54.2	64.4	65.1
C <sub>14</sub> H <sub>10</sub>	4.4	15.5	31.0	49.1	61.4	90.7	92.9
C <sub>14</sub> H <sub>16</sub>	6.3	24.1	43.1	65.0	103.4	129.2	163.7
C <sub>20</sub>	10.5	43.2	102.0	175.7	162.6	233.9	277.5
C <sub>16</sub> H <sub>18</sub>	10.0	38.9	84.1	117.5	192.4	267.9	345.8
C <sub>18</sub> H <sub>12</sub>	14.1	57.1	116.2	178.6	216.4	304.5	380.0
C <sub>18</sub> H <sub>20</sub>	22.5	95.9	161.4	216.3	306.9	512.0	641.3

Statically distribute most diagrams between GPU and CPU,  
dynamically distribute leftovers.

# More hybrid CCSD

molecule	Basis	o	v	Iteration time (s)			Speedup	
				Hybrid	CPU	Molpro	CPU	Molpro
CH <sub>3</sub> OH	aTZ	7	175	2.5	4.5	2.8	1.8	1.1
benzene	aDZ	15	171	5.1	14.7	17.4	2.9	3.4
C <sub>2</sub> H <sub>6</sub> SO <sub>4</sub>	aDZ	23	167	9.0	33.2	31.2	3.7	3.5
C <sub>10</sub> H <sub>12</sub>	DZ	26	164	10.7	39.5	56.8	3.7	5.3
C <sub>10</sub> H <sub>12</sub>	6-31G	26	78	1.4	4.1	7.2	2.9	5.1

Calculations are small because we are not using out-of-core or distributed storage, hence are limited by CPU main memory.

Physics- or array-based domain decomposition will lead to single-node tasks of this size.

# Lessons learned

- **Do not GPU-ize legacy code!**  
Must redesign and reimplement (hopefully automatically).
- Verification is a pain.
- CC possess significant task-based parallelism.
- Threading ameliorates memory capacity and BW bottlenecks.  
(How many cores required to saturate STREAM BW?)
- GEMM and PERMUTE kernels both data-parallel, readily parallelizable via OpenMP or CUDA.
- Careful organization of asynchronous data movement hides entire PCI transfer cost for non-trivial problems.
- Naïve data movement leads to 2x for CCSD; smart data movement leads to 8x.

# Summary of GPU CC

- Implemented CCD on GPU and on CPU using CUDA/OpenMP and vendor BLAS.  
Implementation quality is very similar.
- Implemented CCSD on CPU+GPU using streams and mild dynamic load-balancing.
- Compared to legacy codes *as directly as possible*:
  - Apples-to-apples CPU vs. GPU is 4-5x (as predicted).
  - Apples-to-oranges us versus them shows 7-10x.  
Our CPU code is 2x, so again 4-5x is from GPU.

We have very preliminary MPI results using task parallelism plus `MPI_Allreduce`.

Load-balancing is the only significant barrier to GA+GPU implementation.

# Chemistry Details

Molecule	$o$	$v$
C <sub>8</sub> H <sub>10</sub>	21	63
C <sub>10</sub> H <sub>8</sub>	24	72
C <sub>10</sub> H <sub>12</sub>	26	78
C <sub>12</sub> H <sub>14</sub>	31	93
C <sub>14</sub> H <sub>10</sub>	33	99
C <sub>14</sub> H <sub>16</sub>	36	108
C <sub>20</sub>	40	120
C <sub>16</sub> H <sub>18</sub>	41	123
C <sub>18</sub> H <sub>12</sub>	42	126
C <sub>18</sub> H <sub>20</sub>	46	138

- 6-31G basis set

- C<sub>1</sub> symmetry

- $F$  and  $V$  from GAMESS via disk

Since January ...

- Integrated with PSI3 (GPL).

- No longer memory-limited by GPU.

- Working on GA-like one-sided.

- GPU one-sided R&D since 2009.



# Numerical Precision versus Performance

molecule	Iteration time in seconds					
	C1060		C2050		X5550	
	SP	DP	SP	DP	SP	DP
C <sub>8</sub> H <sub>10</sub>	0.2	0.8	0.2	0.3	0.7	1.3
C <sub>10</sub> H <sub>8</sub>	0.4	1.5	0.2	0.5	1.3	2.5
C <sub>10</sub> H <sub>12</sub>	0.7	2.5	0.4	0.8	2.0	3.5
C <sub>12</sub> H <sub>14</sub>	1.8	7.1	1.0	2.0	5.6	10.0
C <sub>14</sub> H <sub>10</sub>	2.6	10.2	1.5	2.7	8.4	13.9
C <sub>14</sub> H <sub>16</sub>	4.1	16.7	2.4	4.5	12.1	21.6
C <sub>20</sub>	6.7	29.9	4.1	8.8	22.3	40.3
C <sub>16</sub> H <sub>18</sub>	9.0	35.9	5.0	10.5	28.8	50.2
C <sub>18</sub> H <sub>12</sub>	10.1	42.2	5.6	12.7	29.4	50.3
C <sub>18</sub> H <sub>20</sub>	17.2	73.0	10.1	20.1	47.0	86.6

This the apples-to-apples CPU vs. GPU.